华南理工大学
South China University of Technology

课程设计报告书

作业 2：基于神经网络的手写数字识别

学　院　　自动化科学与工程

专　业　　智能科学与技术

学生姓名　田炜滨

学生学号　201830431136

指导教师　李彬

课程编号　046101591

课程学分　2.0

起始日期　2021 年 6 月 9 日

教师评语	教师签名： 日期：
成绩评定	
备注	

基于神经网络的手写数字识别

一、选题背景以及实验目的

1.1 手写字符图像数据集分类

为了实验的一致性和方便对比，作业 2 我们依旧进行手写数字识别，并与支持向量机的作对比和改进。其中手写字符图像每张图像的大小均为 900 *

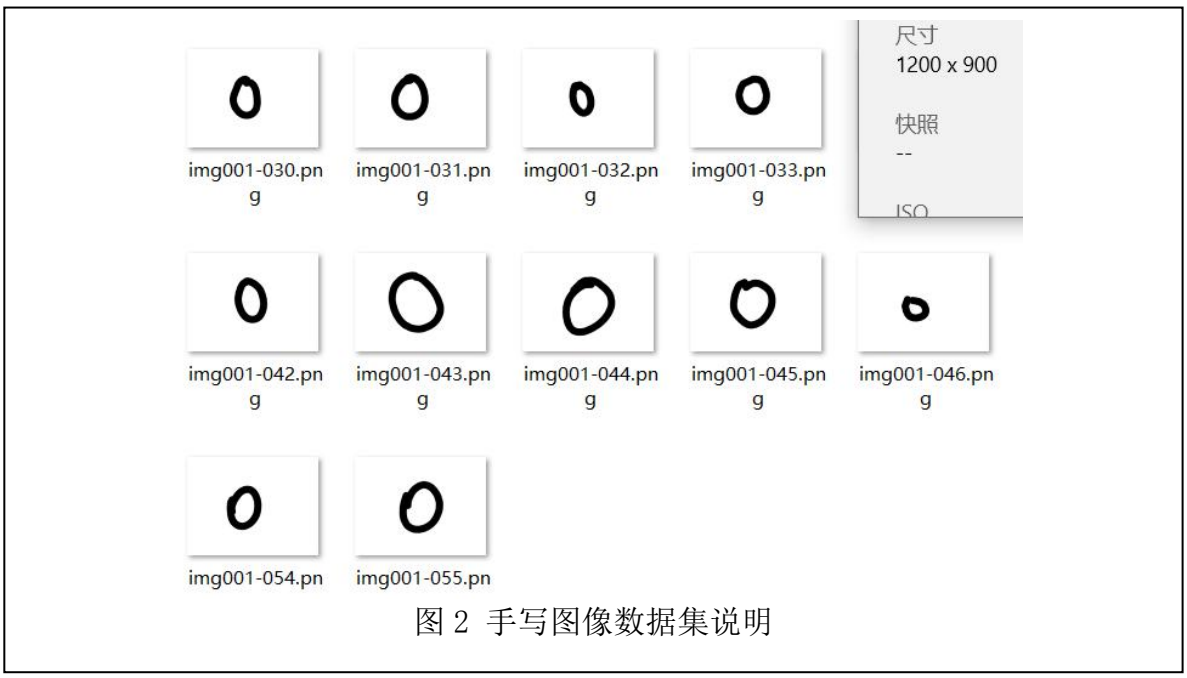


图 2 手写图像数据集说明

1200 ，一共有 10 个类别（即 0-9），每个类别有 55 张图像，即一共有 550 张图像。

对其进行分类时，可视为样本总数为 550，特征向量维数为 900*1200=1080000，目标种类为 10 种的分类问题。特征向量维数太大，不便于直接使用分类模型，而且常用的降维方法在降维运算时都需要巨大的运算消耗。观察图像特征后发现图像存在大量的无效冗余信息，有效特征大多集中在图像中心，可以使用图像缩放的方法，在读取图像信息时先进行图像预处理第一次对数据降维，**裁剪无效特征**信息。之后再使用**主成分分析法**进行第二次降维，保留训练集方差 95% 所需的最小维数从而确定需要降低的最优维数。然后通向使用**神经网络**算法，参数的选择同样使用交叉验证来寻优。最后同样在上述分类器组合使用**投票分类器**以期提高分类器准确率。

1.2 神经网络方法与投票分类器简介

人工神经网络（Artificial Neural Networks，简写为 ANNs）也简称为神经网络或称作连接模型，它是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。人工神经网络按其模型结构大体可以分为前馈型网络（也称为多层感知机网络）和反馈型网络（也称为 Hopfield 网络）两大类，前者在数学上可以看作是一类大规模的非线性映射系统，后者则是一类大规模的非线性动力学系统。按照学习方式，人工神经网络又可分为有监督学习、非监督和半监督学习三类；按工作方式则可分为确定性和随机性两类；按时间特性还可分为连续型或离散型两类，等等。

仅使用神经网络可能效果不是完美，作为扩充，我们加入了之前的 SVM 的分类器作为投票分类器，来作为探索和比对。一堆弱学习模型有时候单个的预测准确率仅仅比随机概率要大一些,那么我们可以根据所有预测器的预测结果占比来输出最终的预测结果。大部分时候基于多预测器的集成学习要优于单个预测,这便是投票器学习。

1.3 实验目的

使用 python 语言，借助 ski-learn 包，使用支持向量机搭建手写数字识别系统，并对实验结果作分析。然后使用 matlab 语言手写反馈神经网络（BP），选择合适的优化器，尝试对上述手写数字再次进行分类。最后结合这两种方法，探索使用其组合的投票方法能否收获更好的结果。

二、实验方法

2.1 手写图像数据集

手写数字识别与酒的识别最大的不同是前者的数据形式是图片，因此需要一定的图像处理知识，图片原尺寸过大，且有用信息少，经图像处理后变为大小适宜的黑白图，再结合 MNIST 数据集提供的一些额外数据，使用神经网络方法完美的得到了高性能分类系统。

神经网络是典型的非线性分类器，它的特点是需要训练数据来确定神经元之间的参数，因此很适合图片识别的问题。

设计神经网络时使用的是 BP 神经网络算法，虽然对于图像处理来说，最好

的神经网络是卷积神经网络，但因为手写数字的图片较为简单，为降低复杂性，使用普通的 BP 神经网络。按照神经网络的知识，搭建输入层、隐藏层和输出层，将数据分为训练集和测试集，又因为所给数据过少，从开源 MNIST 数据集上下载了一些补充数据用于训练。

系统最终实现了对训练集数据进行 0-9 的数字分类，并给出收敛的误差变化图以及正确率 confusion 矩阵，来表示具体的分类情况。该系统的运行环境为 MATLAB2014，使用 gpu 加速，性能与迭代次数成反比。

为了达到与之前作对比，使用投票分类器将 SVM 与神经网络综合起来，来更好的体现两种网络的差异以及综合效果。

2.3 程序实现环境

项目编程语言为 matlab 和 python，原因在于这类语言应用于科学计算时的便捷，

1. scipy

提供了各种高级的工具用于进行数据分析。有许多内置的方法用于分组统计、合并数据、数据筛选、以及时间序列操作。项目中主要用于数据格式整理和 csv 文件的读取。

2. NumPy

支持高阶大量的维度数组与矩阵运算。项目中主要用于算法运行时矩阵运算。

3. Matplotlib

数据可视化工具。本项目中用于分类器估测，样本数据分析时的可使化表达。

4. Scikits_Learn

提供多种实现传统机器学习和数据挖掘任务的算法的接口。本项目中用于部分分类器的调用评估，数据预处理。

5. CV2

OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库，实现了图像处理和计算机视觉方面的很多通用算法。本项目中用于图像数据集的读取，图像数据的裁剪。

6. os 库

提供通用的、基本的操作系统交互功能，包含几百个函数分为路径操作、进程管理、环境参数等几类。本项目中用于读取文件夹路径，读取图像文件夹名。

三、过程论述

3.1 分类器原理阐述

3.1.1 主成分分析

表 1 主成分分析参数表

Notation	Description
x_i	原始特征
ε_i	新特征
α_i	线性组合系数
ε	由新特征ε_i组成的向量
A	特征变换矩阵
∑	协方差矩阵

主成分分析实现原理是找到接近数据集分布的超平面，然后将所有的数据都投影到这个超平面上，从数据计算上的理解是从一组特征中计算出一组按重要性从大到小排列的新特征，新特征是原有特征得线性组合，而且互不相关。

在将训练集投影到较低维超平面的关键是选择正确的超平面，使数据在投影平面内保持最大方差，方差越大则数据的特征提取越理想，数据区分度越好。数据上大部分信息集中在较少的几个主成分上，将数据投影到分布分散的平面内，而忽略掉分布集中的平面，就可以实现在降维的同时让新特征代表原数据。

以某数据集为例，数据一共有 178 个样本，每个样本具有 13 个特征，记为$x1,x2…x13$

设新特征ε_i $i = 1,…13$ 是原始数据 13 个特征的线性组合

$$\varepsilon_i = \sum_{j=1}^{13} \alpha_{ij} x_j = \alpha_i^T x$$

其中α_i^T是线性组合系数，令其模为 1 时可以统一ε_i的尺度。

$$\alpha_i^T \alpha_i = 1$$

以矩阵形式表达如下

$$\varepsilon = A^T x$$

ε由新特征ε_i组成的向量，A 是特征变换矩阵，最优的正交变换 A 需要让新特征ε_i的方差达到极值。

对任意一个新特征ε_i

$$\varepsilon_1 = \sum_{j=1}^{13} \alpha_{1j} x_j = \alpha_1^T x$$

其方差为 $Var(\varepsilon_1)$ = E(ε_1^2)-$E(\varepsilon_1)^2$ = $\alpha_1^T \sum \alpha_1$ 其中Σ为协方差矩阵，在约束条件下$\alpha_i^T \alpha_i$ = 1，求最大化ε_1的方差，可以使用拉格朗日乘数法。

即 $f(\alpha_1)$ = $\alpha_1^T \sum \alpha_1 - v(\alpha_i^T \alpha_i - 1)$

对函数求导使之为0得最优解α_1满足条件$\sum \alpha_1 = v\alpha_1$，可以将该式视为Σ的特征方程，$\alpha_1$为对应的特征向量，$v$为对应的特征值。

计算第二个特征时，除了需要满足第一特征的所有要求之外，还需要与第一特征不相关，满足条件：

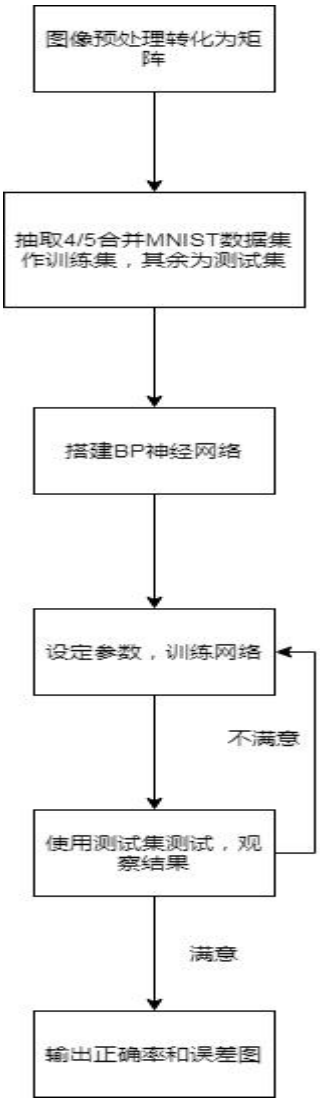
$$E(\varepsilon_2 \varepsilon_1) - E(\varepsilon_2)E(\varepsilon_1) = 0$$

以此类推，求其余特征向量，最后使新特征所能代表的数据总方差比例为95% 确定最终需要降至几维。

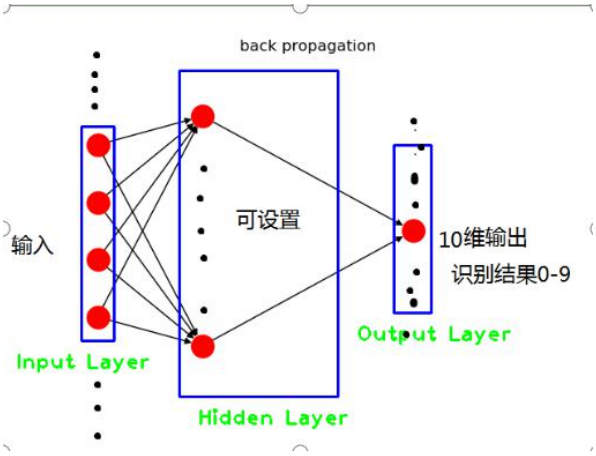
但是上述方法并不利于编程实现，所以我在编程实现时采用的时计算一致性矩阵求解特征值对应的特征向量的办法实现PCA算法。

3.1.3 反馈神经网络算法

算法的思路可用下图表示



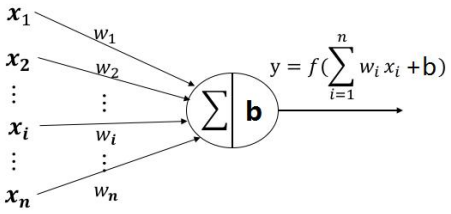
该 BP 神经网络为三层，示意图如下。其中输入层神经元为降维后的维度，数值为图像转化的矩阵。输入是降维后向量，每个分量为图像在对应位置的灰度。隐藏层神经元的个数手动输入可根据情况进行调整，最后输出层为分类结果。即 0-9 数字。



神经网络的设计主要是每一层感知器的输入和输出，即输出 y 和输入 x 的关

系可以用以下映射实现

$$y = f(w * x + b)$$



其中 w 是输入和输出之间的连接权值矩阵（w1,w2,..,wn），x 为输入矩阵(x1,x2,...,xn),b 为偏置，f（●）为激活函数，本设计中使用 Sigmoid 函数，

$$f(\alpha) = \frac{1}{1+e^{-\alpha}}$$

优化器的选择：**梯度下降的优化器**

训练中，因为 BP 算法的特点，需要加入反向传播环节作为优化器对矩阵进行优化，对输出层使用如下公式调节权值

误差的反向传播，即首先由输出层开始逐层计算各层神经元的输出误差，然后根据误差梯度下降法来调节各层的权值和阈值，使修改后的网络的最终输出能接近期望值。Tk 为预期输出

对于每一个样本 p 的二次型误差准则函数为 Ep：

$$E_p = \frac{1}{2}\sum_{k=1}^{L}(T_k - o_k)^2$$

权值修正为

$$\Delta w_{ki} = \eta\sum_{p=1}^{P}\sum_{k=1}^{L}(T_k^p - o_k^p)\cdot\psi'(net_k)\cdot y_i$$

$$\Delta a_k = \eta\sum_{p=1}^{P}\sum_{k=1}^{L}(T_k^p - o_k^p)\cdot\psi'(net_k)$$

$$\Delta w_{ij} = \eta\sum_{p=1}^{P}\sum_{k=1}^{L}(T_k^p - o_k^p)\cdot\psi'(net_k)\cdot w_{ki}\cdot\phi'(net_i)\cdot x_j$$

$$\Delta\theta_i = \eta\sum_{p=1}^{P}\sum_{k=1}^{L}(T_k^p - o_k^p)\cdot\psi'(net_k)\cdot w_{ki}\cdot\phi'(net_i)$$

其中 w 指权值，a 与θ分别为输出层和隐含层偏置，将该步长加到原来对应的值即为更新后的值。这种更新权值方法使用的矩阵优化器为梯度下降优化器，

即按每个θ的负梯度方向对权值进行更新。

$$\theta_{i+1}' = \theta_i - \Delta\theta_i$$

每个类选 5 张一共 50 张作为测试集，测试集需要使用标签来验证结果，标签为 50*10 的矩阵，数字 1 所在的位置决定该图片的数字类型。最后计算正确率和误差，以及输出 confusion 矩阵作为评价算法好坏的标准。

3.1.4 集成学习-投票分类器（用于组合和对比）

聚合每个分类器预测的类别，然后选其中投票最多的类别，这种多数投票分类器称为一个硬投票（hard voting）分类器。

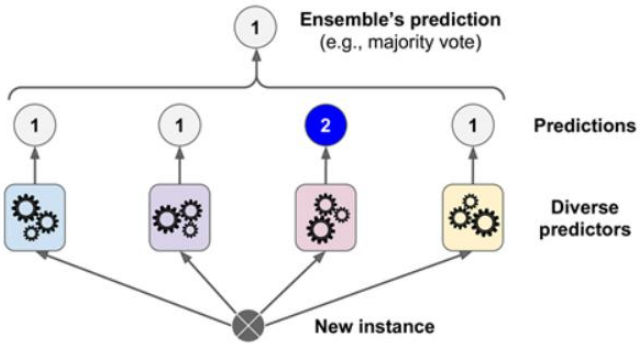


图 3 投票分类器原理

即使每个分类器都是一个弱学习者（weak learner，也就是说它的预测能力仅比随机猜稍微高一点），集成的结果仍可以是一个强学习者（strong learner，能达到高准确率）

成立的条件在于：所有的分类器都是完全独立的，它们产生的错误也都是不相关的。若都是在同一个数据集上进行的训练，它们更有可能产生同样类型的错误，所以也会有很多票投给错误的类别，并最终降低了集成的准确率。集成学习在预测器互相之间（尽可能地）独立时表现最好，我的实现办法就是使用完全不同的算法训练不同的分类器，这个可以增加它们产生不同类型错误的几率，提升集成的准确度。

3.2 算法实现流程

3.2.1 手写图像数据集

数据读取与格式整理

由于图像信息只有黑白两种颜色，所以我采用了灰度形式读取图像信息。

```
1.  for label in os.listdir('./Img'):
2.      for filename in os.listdir('./Img'+"/"+label):
3.          img = cv2.imread('./Img'+"/"+label + "/" + filename,0) # 以灰度形式读取图片
```

图像数据集的读取使用 cv2 与 os 库中封装函数 os.listdir() 与 cv2.imread()，将图像以灰度的形式读取进来，数据样本规模为 550（0-9 一共十种类别，每种类别各有 55 张图片），单张图片信息为 900*1200 大小的矩阵，也可以将其视为初始特征向量为（1*1080000）维，而且有大量的属性值为无效特征，可以从中看出，在将数据代入分类器分类前，需要先进行特征提取实现降维。但是在实现特征提取的过程中，以 PCA 为例，同样需要大量矩阵运算，数据运算时的大量依旧无法避免，所以我们还需要在实现降维之前先对数据进行无效信息的裁剪。

数据预处理（第一次降维）

进行无效信息裁剪时，先观察图像数据特点，发现图像边缘都是无效的白色背景特征，数字在图像中基本占据中心位置，可以通过图像缩放降低数据维数。

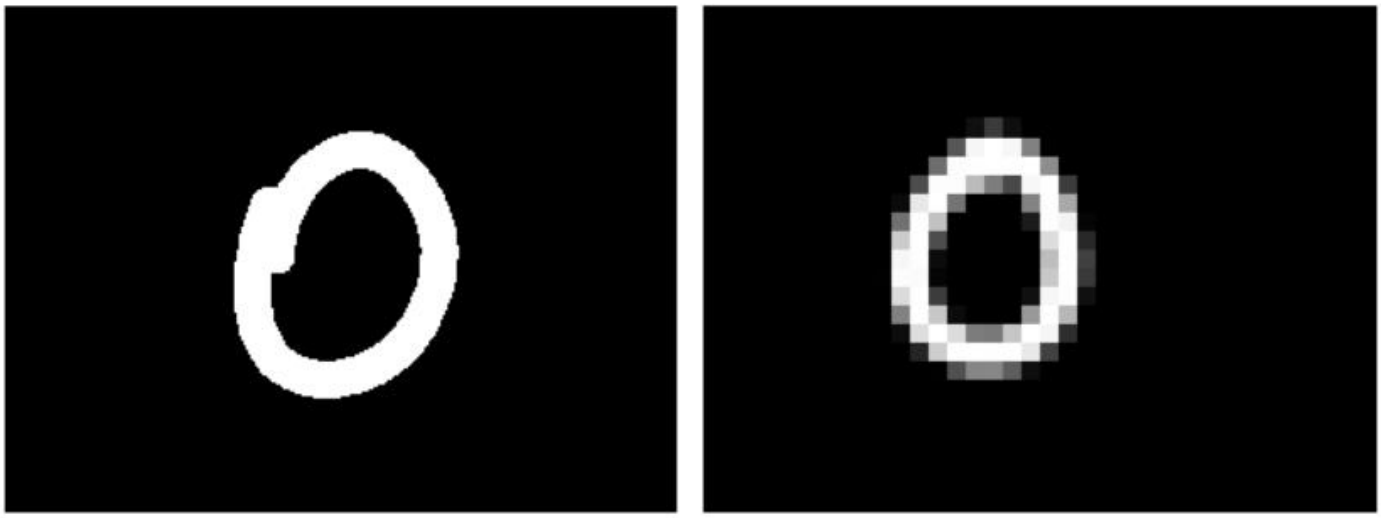


图 5 缩放降维前后图像对比图

如图可以看出，尽管维数有较大下降，但是图像依旧特征较为明显，可以实现避免大量数据运算的同时保持图像特征，使属性维数下降。此时特征维数为（1*972）维。

主成分分析（第二次降维）

使用主成分分析降低数据维数时，需要确定需要降低的目标维数，我计算出保留训练集方差 95% 所需的最小维数,作为其最终要下降的目标维数。

```
from sklearn.decomposition import PCA
# 在不降维的情况下进行 PCA，然后计算出保留训练集方差 95% 所需的最小维数
pca=PCA()
pca.fit(data)
cumsum=np.cumsum(pca.explained_variance_ratio_)
d=np.argmax(cumsum>=0.95)+1
```

绘制方差解释率关于维数的函数，保留训练集方差 95% 所需的最小维数时，选择最优的维数是 86 维，所以将此时特征维数为（1*972）维的特征向量降至（1*86）维。

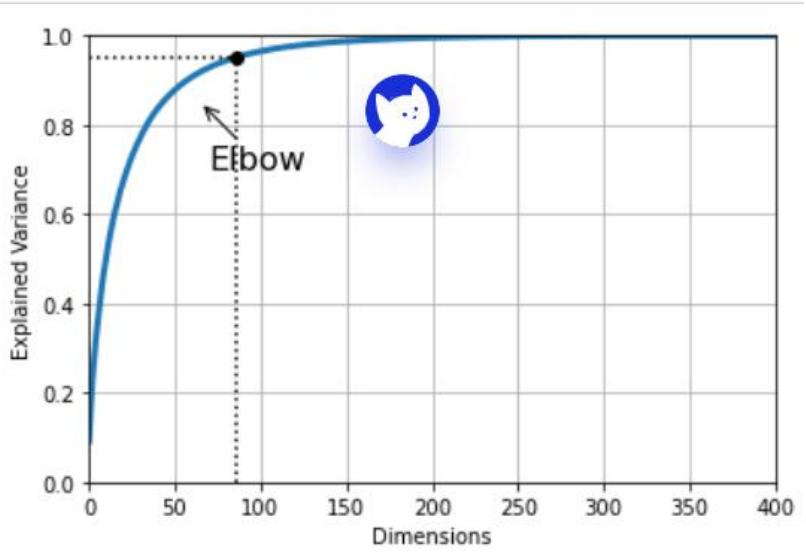


图 6 方差解释率关于维数的函数

如上图所示，当维数低于 86 维时，训练集方差解释率将低于 95%，不利于后续的数据分类，当高于 86 维时，曲线上升平缓，维数增加带来的方差解释率的增长较小，而且维数增加意味着后续分类器的运算规模的增加，所以维数不宜过大，综上所述维数选取 86 维比较合适。

为了观察其降维效果，再调用 data_inverse（）将降至 86 维后的特征向量重新恢复为（27*36）972 维观察图像，对比二者可得。

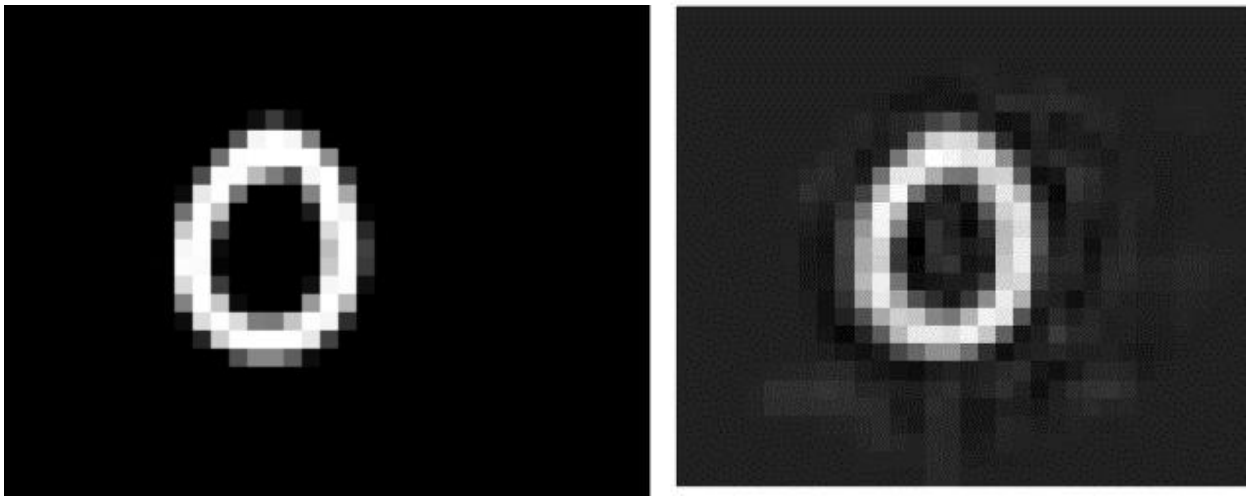


图 7 PCA 降维前与 PCA 降维后（经过恢复再以图像形式输出）图像对比图

如图可以看出，经过 PCA 降维之后再恢复，图像信息变得更加模糊，说明存在有效特征的丢失，但是主要特征依旧保存良好，证明了两次降维方法的可行性。

确定目标维数之后，PCA 降维的具体算法可参考 3.2.1 wine 数据集 的主成分分析部分。

测试集与训练集划分

同样由于受限于原始数据总量规模较小，所以没有划分验证集，仅划分了测试集和训练集，且二者占总数据比列为 2：8.原始数据 10 个类别（0-9）各自占样本总数的比列相同，故测试集和训练集中同样取各类别占比相同。

```
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.2,random_state=4,stratify=target)
print(x_train.shape)
print(y_train.shape)
```

数据标准化

由于图像数据集的特殊性，灰度形式读取后所有的属性值都在 0-255 范围内，且原图片为黑白双色，所以数据的特征值本身就可以在数值上进行比较，没有再次数据标准化的必要。另一方面，经过实验测试发现在不对数据进行数据标准化时分类器的正确率更高。

反馈神经网络分类器

函数实现：

```
function [y] = layerout(w,b,x)
% 函数作用：1 针对每个图像样本计算隐层神经元的输出，其激活函数为 S 函数；
```

```
%  2 根据隐层神经元输出值计算输出层的输出；
% x 是一列数据 784 个数据，代表一幅图
% w 是输入层到隐层权值矩阵 n 行，784 列
% y 是输出层的各神经元输出，是一个 10 层的列向量，输出神经元数 10
y = w*x + b;                            % y 是 n 个隐层神经元的输出，也是列向量
n = length(y);                          % n 隐层神经元数
for i =1:n
    y(i)=1.0/(1+exp(-y(i))); % 隐层神经元利用 S 函数计算输出
end
y;
end
        %更新公式的实现，典型的反向传播算法更新各权值
        o_update = (y-out_put).*out_put.*(1-out_put);
        h_update = ((w')*o_update).*hid_put.*(1-hid_put);

        outw_update = a*(o_update*(hid_put'));%输出层权值
        outb_update = a*o_update;%输出层偏置
        hidw_update = a*(h_update*(x'));%隐藏层权值
        hidb_update = a*h_update;%隐藏层偏置
        w = w + outw_update;
      b = b+ outb_update;
      w_h = w_h +hidw_update;
      b_h =b_h +hidb_update;%更新
用户控制输入
step = input('迭代步数：');
a = input('学习因子：');
in = 784;                   % 输入神经元个数
hid = input('隐藏层神经元个数：');    % 隐藏层神经元个数 n
out = 10;               % 输出层神经元个数
```

多分类器组合的投票分类器

投票分类器用于将多个弱分类器组合进行投票，当分类器的类型不同时往往才有较为理想的效果，为了与作业一形成对比，并进一步改善其性能，使用该投票分类器采用的是 SVM(kernel ='rbf'),神经网络分类器的组合。

```
y_predict_voting_all = np.vstack(( y_predict_knn,  y_predict_svm, y_predict_bp))
print(y_predict_voting_all)
from scipy import stats
y_predict_voting_all = stats.mode(y_predict_voting_all)[0][0] # 取众数
print(y_predict_voting_all)
print(type(y_predict_voting_all))
print(( y_test == y_predict_voting_all).sum()/len(y_test))
print(metrics.classification_report(y_test, y_predict_voting_all))
print(metrics.confusion_matrix(y_test,  y_predict_voting_all))
```

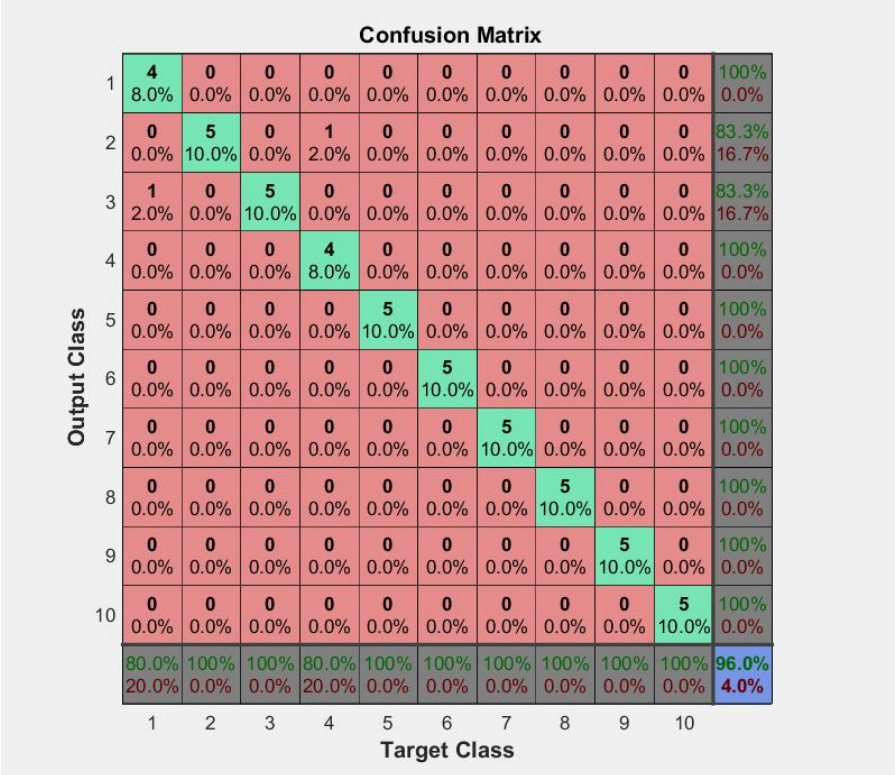
四、结果分析

4.1 手写图像数据集神经网络分类结果

手写数字识别结果的好坏关键在于训练时所给的参数以及选择的优化器，在经过不断的调整参数（炼丹）过程后，我得到了目前为止效果最好的参数为（学习因子就是梯度优化器每次梯度下降中的步长）

```
迭代步数：1500
学习因子：0.02
隐藏层神经元个数：24
```

详细测试结果（共 50 张图片）



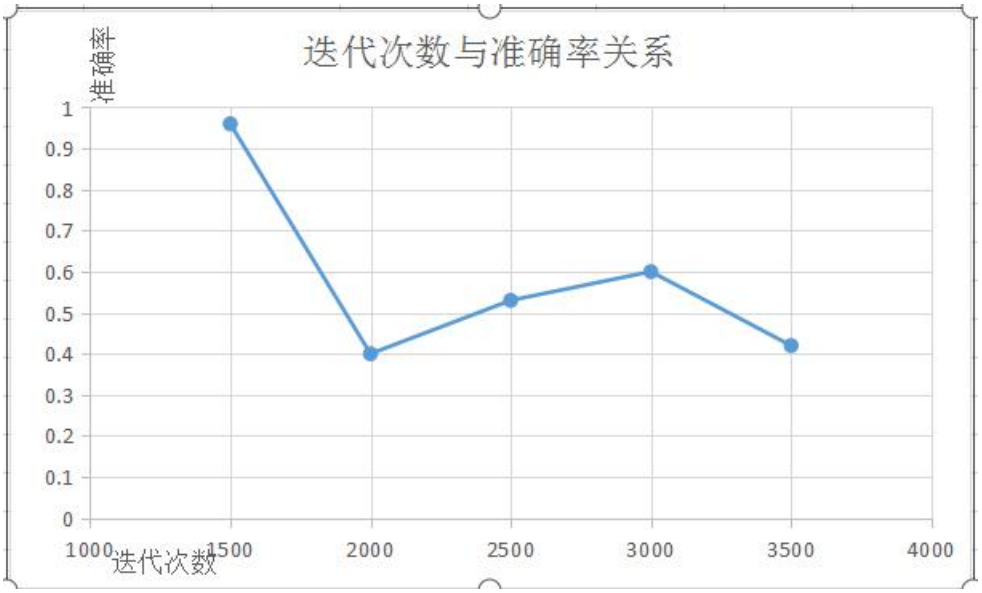
分类器	反馈神经网络
参数	hidden_layer_sizes=(900, 1500)
正确率	0.936363636

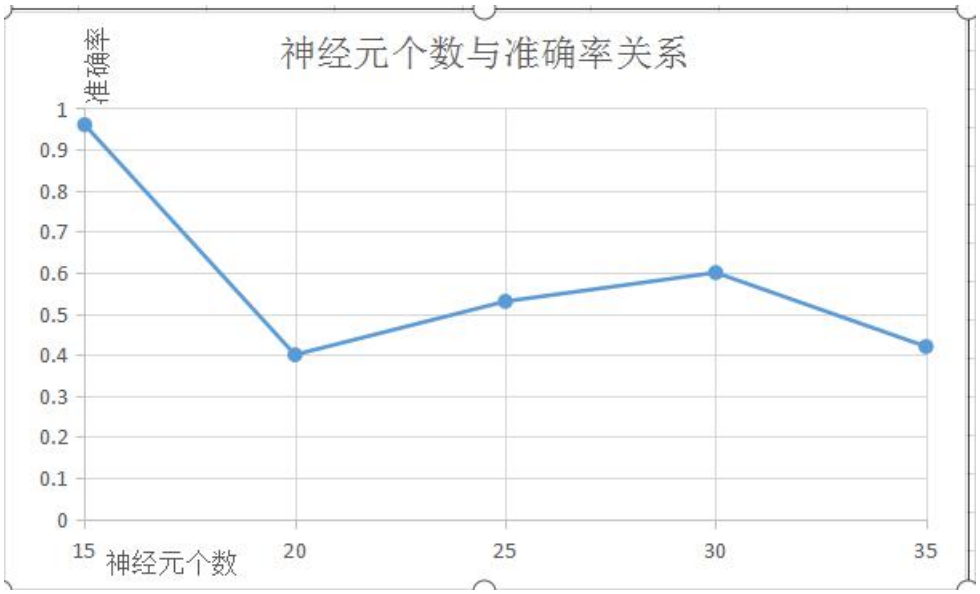
混淆矩阵	[[22 0 0 0 0 0 0 0 0 0] [0 22 0 0 0 0 0 0 0 0] [0 0 18 4 0 0 0 0 0 0] [0 0 0 22 0 0 0 0 0 0] [0 0 2 0 18 0 2 0 0 0] [0 0 0 2 0 16 4 0 0 0] [0 0 0 0 0 0 22 0 0 0] [0 0 0 0 0 0 0 22 0 0] [0 0 0 0 0 0 0 0 22 0] [0 0 0 0 0 0 0 0 0 22]]

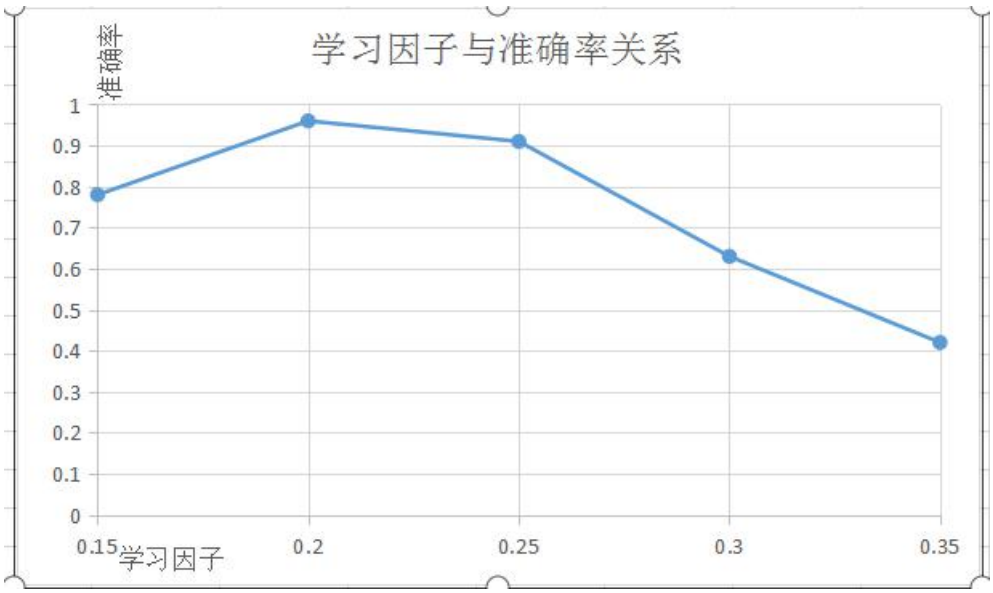
具体结果（confusion 图为测试集结果，表为 MNIST 数据集结果）

Confusion 图中横轴是预计分类（正确分类），1-10 类依次代表 0-9 数字，纵轴是实际分类，其中分类正确的图片数量位于绿色对角线上，灰色是准确率。从图片上我们可以看出，小样本分类器只分错了 2 张图片，准确率高达 96%，可以说出色的完成了分类任务。而大规模 MNIST 样本的正确率也高达 93%。

我随后单独改变迭代次数、神经元个数和学习因子，发现他们与准确率关系为







可见参数对准确率的影响很大，但规律并不是很明显。这是由于神经网络非线性的特点。细微参数的变化往往会导致结果的巨大改变，因此如果想进一步的提升准确率，必须改善网络结构。

4.3 手写图像数据集神经网络+SVM 投票分类结果

与实验一单纯的 SVM 作对比得到的结果为下表

分类器	SVM	反馈神经网络	KNN（用作参照）
参数	C = 2.0 , kernel ='rbf'	hidden_layer_sizes=(900, 1500)	n_neighbors=1
正确率	0.940909091	0.936363636	0.954545455
混淆矩阵	[[22 0 0 0 0 0 0 0 0 0] [0 22 0 0 0 0 0 0 0 0] [2 3 17 0 0 0 0 0 0 0] [0 0 0 22 0 0 0 0 0 0] [0 0 0 0 20 0 2 0 0 0] [0 0 0 3 0 19 0 0 0 0] [0 0 0 0 1 0 21 0 0 0] [0 1 0 0 0 0 0 21 0 0] [0 0 0 0 0 0 0 0 22 0] [0 0 0 0 1 0 0 0 0 21]]	[[22 0 0 0 0 0 0 0 0 0] [0 22 0 0 0 0 0 0 0 0] [0 0 18 4 0 0 0 0 0 0] [0 0 0 22 0 0 0 0 0 0] [0 0 2 0 18 0 2 0 0 0] [0 0 0 2 0 16 4 0 0 0] [0 0 0 0 0 0 22 0 0 0] [0 0 0 0 0 0 0 22 0 0] [0 0 0 0 0 0 0 0 22 0] [0 0 0 0 0 0 0 0 0 22]]	[[22 0 0 0 0 0 0 0 0 0] [0 22 0 0 0 0 0 0 0 0] [0 0 20 0 0 0 0 2 0 0] [0 0 0 22 0 0 0 0 0 0] [0 0 0 0 20 0 0 2 0 0] [0 0 0 2 0 20 0 0 0 0] [0 0 0 0 0 0 20 0 2 0] [0 0 0 0 0 0 0 22 0 0] [0 0 0 0 0 0 0 0 20 2] [0 0 0 0 0 0 0 0 0 22]]
时间 s	144.2	89.4	75.4

由表中结果可得神经网络在准确率上低于 SVM，但差距不是很大，因为同样都是非线性结构，准确性本质上差距不大。但在性能时间消耗上，因为 SVM 采用的是 python 自带的包，运算时更为复杂，而神经网络是自己写的，所以结构更加简单，因此速度也更快。实际使用时，应该尽量使用自己手写的代码。

表 9 图像数据集投票分类器分类结果

分类器	投票分类器
参数	y_predict_knn, y_predict_svm, y_predict_bp
正确率	0.972727273

混淆矩阵	[[22 0 0 0 0 0 0 0 0 0] [0 22 0 0 0 0 0 0 0 0] [0 2 20 0 0 0 0 0 0 0] [0 0 0 22 0 0 0 0 0 0] [0 0 0 0 20 0 2 0 0 0] [0 0 0 2 0 20 0 0 0 0] [0 0 0 0 0 0 22 0 0 0] [0 0 0 0 0 0 0 22 0 0] [0 0 0 0 0 0 0 0 22 0] [0 0 0 0 0 0 0 0 0 22]]
时间 s	82.3

对于手写图像数据集数据集，图像第一次裁剪降维至（27*36）972 维，第二次降维使用 PCA 降维至 86 维，利用集成学习思想使用投票分类器后可以使准确率达到 97.2%，多次测试发现分类效果理想。由此可见，当使用组合投票分类器时，可以对分类的效果大大提升。

五、课程设计总结

本次课程设计在具体情景下完成了分类器分类全部过程。在将理论算法流程转化为实际可行的代码过程中，遇到了很多偏向实践的问题，例如：

1. PCA 算法原理上是将原特征线性组合成一组新的互不相关的特征向量，实际代码实现利用的是协方差矩阵法；
2. 图像数据集进行特征提取时要尽可能避免运算消耗等。对于图像数据集可以考虑无效背景裁剪，对于语音数据集可以考虑噪音信号的滤波。
3. 考虑到神经网络的特性，详细讨论了该分类器在不同参数下，不同神经元个数以及不同优化器步长的分类效果，并对过拟合现象做出分析。
4. 集成学习的思想对于解决实际问题很有帮助，分类器分类效果受分类器种类的选择较大，尝试不同种类的分类器，然后利用类型差异较大的分类器组成投票分类器提高准确率的方法实现简单，具有可行性。

5. 与作业一的 SVM 支持向量机方法作对比，初步结论是神经网络泛化能力更强，时间短，但准确率较低。

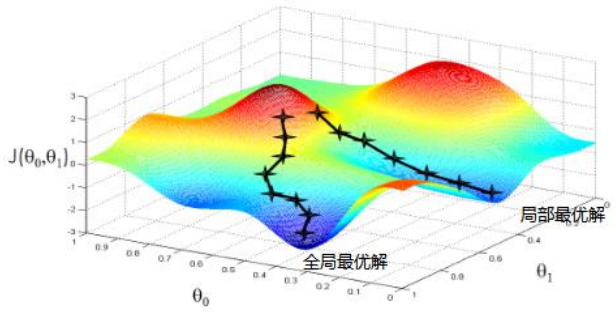
六、对一些问题的探索与思考

6.1 优化器的相关问题

对于优化器的选择，本文使用的是在前面我们实现的神经网络中所使用的优化方法是**随机梯度下降法**（Stachastic gradient desent 简称 SGD）。SGD 的想法就是沿着梯度的方向前进一定距离。用数学的语言来描述的话可以写成下式：

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

当函数呈延伸状的情况下，梯度指向了谷底，而不是直接指向了最低点，函数值在学习过程中会来回震荡，但是向最低点移动的却很小。从数学角度看，传统的 BP 神经网络为一种局部搜索的优化方法，它要解决的是一个复杂非线性化问题，网络的权值是通过沿局部改善的方向逐渐进行调整的，这样会使算法陷入局部极值，权值收敛到局部极小点，从而导致网络训练失败。加上 BP 神经网络对初始网络权重非常敏感，以不同的权重初始化网络，其往往会收敛于不同的局部极小，这也是我每次训练得到不同结果的根本原因。



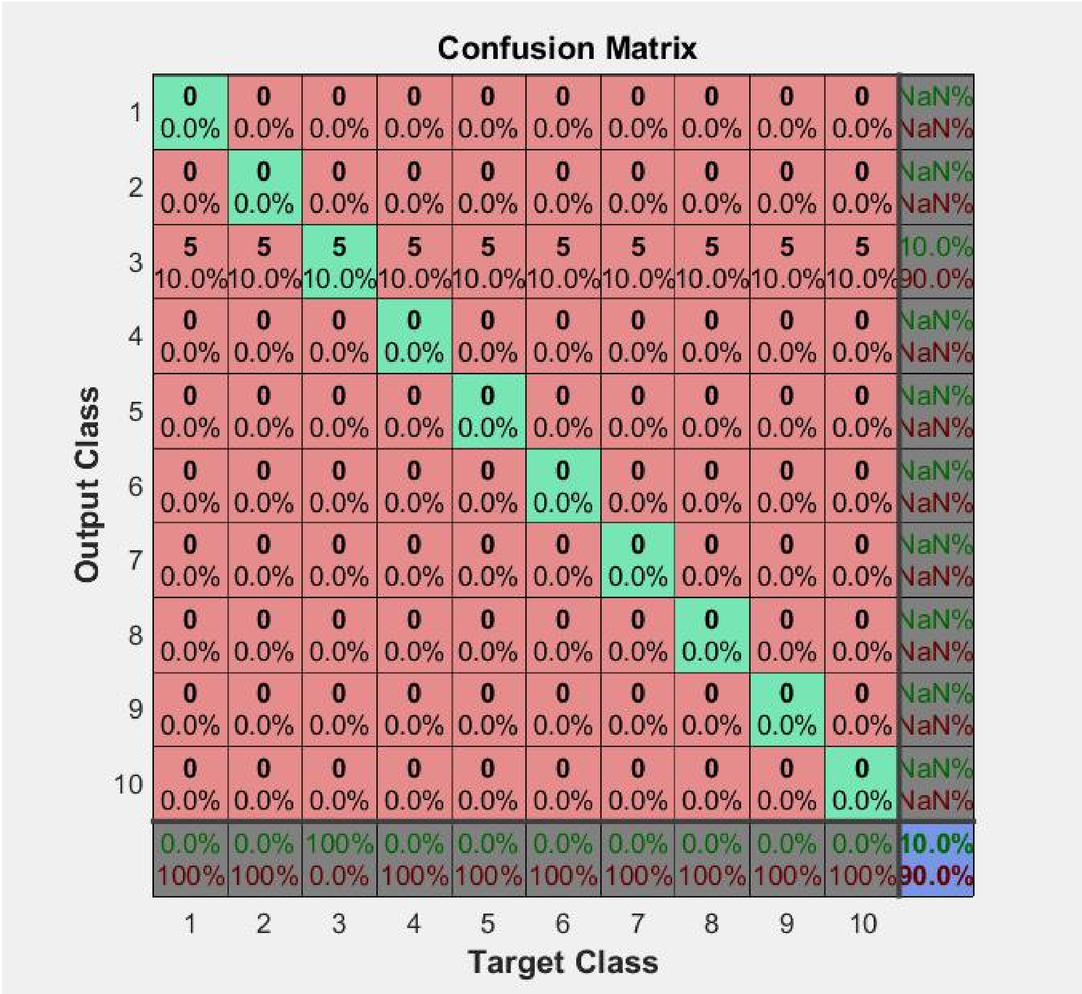
局部最优化示意图

例如，在局部最优化的影响下，分类器很可能认为 2 和 7 是一类数，因为他们的轨迹很相似，虽然 2 比 7 多一横，但拟合计算误差时很可能发现 2 与 7 的其他特征吻合的很好，从而忽略了下面的横，将 2 与 7 的误差进行反向传播，从而陷入局部最优，将 2 和 7 归为一类。

如果学习因子过小，隐藏层神经元过多的话，则会遇到局部最优化问题。例如，我某次训练使用的参数

```
迭代步数：2000
学习因子：0.001
隐藏层神经元个数：64
```

相比最好参数，这组参数的值都大很多，按理说效果会好很多，然而



结果正确率为 10%！而且所有图片都被识别为 2，这一问题的原因是局部最优解，局部最优解即某个范围内的最优解，但不是全局的，从而导致结果“陷”在某个极值中。

解决方法：

局部最优化带来的过拟合问题，与神经元个数和学习因子直接相关，当学习因子过小时，每次更新的步长也会小，容易造成输出值与真实值的误差变化较小，使得神经元参数不再更新，“陷”入局部极值。解决方法可以采用模拟退火方法，每次的更新步长加一个随机数，即

$$P(dE) = \exp\left(\frac{dE}{kT}\right)$$

6.2 GPU 加速

GPU 相比 CPU 更适合神经网络训练等过程，matlab 含有支持 GPU 的语句，使用方法为

```
A_gpu = gpuArray(A);
```

生成一个 gpu 数组，使用英伟达显卡进行加速后，效果非常明显，性能成倍

增加。

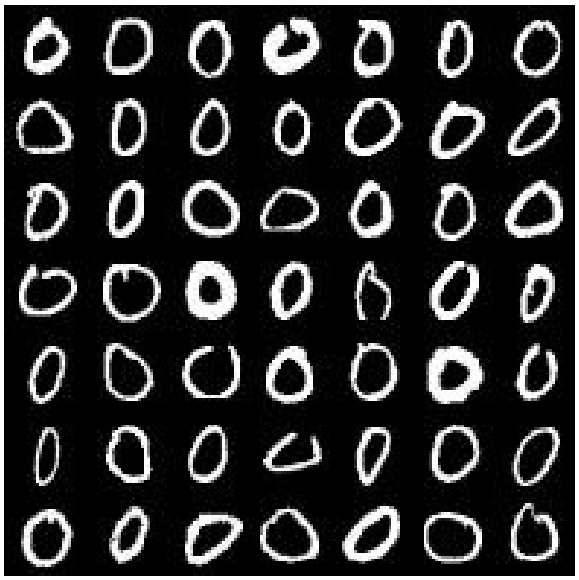
网络鲁棒性的思考:

对抗样本的存在表明现代神经网络是相当脆弱的。所谓对抗性样本，就是在图片上叠加了一些噪声，从人的识别看是没有区别的，但对计算机却是完全两种图像。在网络上我得到了关于鲁棒性的资料。为了检验该思想，我使用 matlab 图像反转，将图片黑白转置，再加入一些噪声，得到示例为

再次进行测试，发现结果差距不是很大，可能是因为网络结构比较简单，对输入数据敏感度并不高。至于其他检测鲁棒性的方法还在探索中。

网络实时性和推广性的思考

在进行手写数字识别的时候，我曾经想过设计一个实时的交互程序，即用户能够自己手写一个数字，软件可以立马识别出该数字的值。但我发现实现过程困难重重，首先 matlab 的应用打包支持并不好，另外最关键的是，网络的实时性很差，输入的数据需要一定时间才能得到结果。而在实际使用中，用户对延迟的忍耐是很低的，且真正的手写识别中，用户写的数字的复杂程度远比题目所给的复杂，例如我在 MNIST 看到一些手写数字集



实际用户输入数据

对于这种数据，本文使用的网络的效果就不是那么明显，因为其同一数字间的形态差异也很大，而且这样计算需要的时间更长，因此可能需要卷积神经网络等专用来进行视觉识别的神经网络。

七、参考文献

[1]周志华.机器学习[M].清华大学出版社:北京,2016:23-246.
[2] 机器学习算法与自然语言处理
https://zhuanlan.zhihu.com/p/77750026
[3] svm.SVC 参数详解
https://blog.csdn.net/weixin_41990278/article/details/93137009
[4]支持向量机(SVM)——SMO 算法
https://zhuanlan.zhihu.com/p/32152421
[5] 机器学习算法原理总结系列---算法基础之支持向量机(Support Vectors Machine)
https://blog.csdn.net/Tong_T/article/details/78918068?utm_source=blogxgwz5
[6] 机器学习-支持向量机(python3 代码实现)
https://blog.csdn.net/weixin_41090915/article/details/79177267
[7] sklearn 中文文档
http://www.scikitlearn.com.cn/
[8] 机器学习之 KNN（k 近邻）算法详解
https://blog.csdn.net/sinat_30353259/article/details/80901746
[9] PCA 算法原理
https://blog.csdn.net/qq_29238153/article/details/80822372
[10] 机器学习（ML）三之多层感知机
https://www.cnblogs.com/jaww/p/12302543.html
[11] 机器学习--集成学习（Ensemble Learning）
https://www.cnblogs.com/zongfa/p/9304353.html